

An Optimal Labeling Scheme for Ancestry Queries*

Pierre Fraigniaud

CNRS and Univ. Paris Diderot
pierre.fraigniaud@liafa.jussieu.fr

Amos Korman

CNRS and Univ. Paris Diderot
amos.korman@liafa.jussieu.fr

Abstract

An *ancestry labeling* scheme assigns labels (bit strings) to the nodes of rooted trees such that ancestry queries between any two nodes in a tree can be answered merely by looking at their corresponding labels. The quality of an ancestry labeling scheme is measured by its label *size*, that is the maximal number of bits in a label of a tree node.

In addition to its theoretical appeal, the design of efficient ancestry labeling schemes is motivated by applications in web search engines. For this purpose, even small improvements in the label size are important. In fact, the literature about this topic is interested in the exact label size rather than just its order of magnitude. As a result, following the proposal of a simple interval-based ancestry scheme with label size $2\log_2 n$ bits (Kannan et al., STOC '88), a considerable amount of work was devoted to improve the bound on the size of a label. The current state of the art upper bound is $\log_2 n + O(\sqrt{\log n})$ bits (Abiteboul et al., SODA '02) which is still far from the known $\log_2 n + \Omega(\log \log n)$ bits lower bound (Alstrup et al., SODA'03).

In this paper we close the gap between the known lower and upper bounds, by constructing an ancestry labeling scheme with label size $\log_2 n + O(\log \log n)$ bits. In addition to the optimal label size, our scheme assigns the labels in linear time and can support any ancestry query in constant time.

*This research is supported in part by the ANR projects ALADDIN and PROSE, and by the INRIA project GANG.

1 Introduction

1.1 Background

In this paper we consider the following problem. Given an n -node rooted tree T , label the nodes of T in the most compact way such that given any pair of nodes u and v , one can determine whether u is an ancestor of v in T by merely inspecting the labels of u and v . The main quality measure used to evaluate such an *ancestry labeling scheme* is the *label size*, that is, the maximum number of bits stored in a label of a node, taken over all nodes in all possible n -node rooted trees.

Among other things, the above elegant problem is not only of fundamental interest but is also useful for performance enhancement of XML search engines. In the context of this application, each indexed document is a tree, and the labels of all trees are maintained in the main memory¹. Therefore, even small improvements in the label size are important, and, in fact, the literature about this topic is interested in the exact label size rather than just its order of magnitude (e.g., label size $\frac{3}{2} \log n$ bits is considered significantly better than label size $2 \log n$ bits²).

Ancestry schemes which are currently being used by actual systems are variants of the following simple interval-based ancestry labeling scheme [16] (see also [18]). Given an n -node tree T , perform a DFS traversal in T starting at the root, and provide each node u with a DFS number $\text{DFS}(u)$ in the range $[0, n - 1]$. Then the label of a node u is simply the interval $I(u) = [\text{DFS}(u), \text{DFS}(v)]$, where v is the descendant of u with largest DFS number. An ancestry query then amounts to an interval containment query between the corresponding labels: a node u is an ancestor of a node v if and only if $I(v) \subset I(u)$. Clearly, the label size of this scheme is $2 \log n$ bits.

An elegant lower bound of $\log n + \Omega(\log \log n)$ bits on the label size is given in [3]. This lower bound holds even for a very restricted family of trees, each composed of equal length simple paths hanging down from the root.

In the other direction, a considerable amount of research has been devoted to improve the upper bound on the label size as much as possible beyond the trivial $2 \log n$ bound [1, 2, 6, 8, 15, 20]. Specifically, [2] gave a first non-trivial upper bound of $\frac{3}{2} \log n + O(\log \log n)$ bits. This was improved the year after to $\log n + O(\sqrt{\log n})$ [6], which is the current best upper bound (that scheme is described in detail in the joint journal publication [1]). In addition to its relatively small label size, the scheme in [6] also assigns labels in linear time and can answer any ancestry query in constant time. Independently of that work, an ancestry labeling scheme with larger label size of $\log n + O(\log n / \log \log n)$ was given in [20].

Following the above results, two other works were published, which focused on particular types of trees. Specifically, an experimental comparison of different ancestry labeling schemes on XML trees that appear in real life can be found in [15]. Recently, [8] gave an ancestry labeling scheme which is efficient for trees of small depth; specifically, for n -node trees with depth d , their scheme uses labels of size $\log n + 2 \log d + O(1)$.

1.2 Our results

In this paper we close the gap between the known lower and upper bounds, by constructing an ancestry labeling scheme for general rooted n -node trees with label size $\log n + O(\log \log n)$. This

¹Details on XML search engines and their relation to ancestry labeling schemes can be found, e.g., [1, 2, 8].

²All logarithms in this paper are taken in base 2.

solves one of the main open problems in the field of informative labeling schemes. In addition to the optimal label size, our scheme assigns the labels to the nodes of a tree in linear time, and guarantees that any ancestry query can be answered in constant time.

1.3 Related work

As explained in [16], the names of nodes in traditional graph representations reveal no information about the graph structure and hence memory is wasted. Moreover, typical representations are usually global in nature, i.e., in order to derive useful information, one must access a global data structure representing the entire network, even if the sought information is local, pertaining to only a few nodes. In contrast, the notion of *informative labeling schemes*, introduced in [16], involves an informative method for assigning labels to nodes. Specifically, the assignment is made in a way that allows one to infer information regarding any two nodes directly from their labels, without using any additional information sources. Hence in essence, this method bases the entire representation on the set of labels alone. This method was illustrated in [16], by giving two elegant and simple labeling schemes for n -node trees: one supporting adjacency queries and the other supporting ancestry queries. Both schemes incur $2 \log n$ label size.

As mentioned earlier, ancestry labeling schemes were further investigated in [1, 3, 2, 6, 8, 15, 20], and the current state of the art upper and lower bounds are $\log n + O(\sqrt{\log n})$ and $\log n + \Omega(\log \log n)$, respectively. Adjacency labeling schemes on trees were also further investigated in an attempt to optimize the label size beyond the simple $2 \log n$ bound of [16]. The current state of the art upper bound [5] for that problem is $\log n + O(\log^* n)$.

Labeling schemes were also proposed for other decision problems on graphs, including distance [3, 10, 19], routing [7, 13, 20], flow [14, 11], vertex connectivity [12, 11], nearest common ancestor [4, 17], and various other tree functions, such as center, separation level, and Steiner weight of a given subset of vertices [17]. See [9] for a partial survey on labeling schemes.

2 Preliminaries

Let T be a tree rooted at some node r referred as the *root* of T . For two nodes u and v in T , we say that u is an *ancestor* of v if $u \neq v$ and u is one of the nodes on the shortest path connecting v and r in T . For every non-root node u , let $\text{parent}(u)$ denote the parent of u , i.e., the ancestor of u at distance 1 from it. A node v is a *descendant* of u if and only if u is an ancestor of v .

The *depth* of a node $u \in V(T)$ is defined as the distance from u to the root of T , i.e., the number of edge traversals from u to the root. In particular, the depth of the root is 0. The *size* of T , denoted by $|T|$, is the number of nodes in T . The *weight* of a node $u \in V(T)$, denoted by $\text{weight}(u)$, is defined as 1 plus the number of descendants of u , i.e., $\text{weight}(u)$ is the size of the subtree hanging down from u . In particular, the weight of the root is $\text{weight}(r) = |T|$. Let $\mathcal{T}(n)$ denote the family of all rooted trees of size at most n .

An *ancestry labeling scheme* $(\mathcal{M}, \mathcal{D})$ for the family of trees $\mathcal{T}(n)$ is composed of the following two components:

1. A *marker* algorithm \mathcal{M} that, given a tree $T \in \mathcal{T}(n)$, assigns labels (i.e., bit strings) to its nodes.

2. A *decoder* algorithm \mathcal{D} that, given any two labels ℓ_1 and ℓ_2 in the output domain of \mathcal{M} , returns a boolean $\mathcal{D}(\ell_1, \ell_2)$.

These components must satisfy that if $L(u)$ and $L(v)$ denote the labels assigned by the marker to two nodes u and v in some rooted tree $T \in \mathcal{T}(n)$, then

$$\mathcal{D}(L(u), L(v)) = 1 \iff u \text{ is an ancestor of } v \text{ in } T.$$

It is important to note that the decoder \mathcal{D} is independent of the tree T . That is, given the labels of two nodes, the decoder decides the ancestry relationship between the corresponding nodes without knowing to which tree in $\mathcal{T}(n)$ they belong to.

The common complexity measure used to evaluate the quality of an ancestry labeling scheme $(\mathcal{M}, \mathcal{D})$ is the *label size*, that is the maximum number of bits in a label assigned by the marker algorithm \mathcal{M} to any node in any tree $T \in \mathcal{T}(n)$.

When considering the *query time* of the decoder, we use the RAM model of computation, and assume that the length of a computer word is $\Omega(\log n)$ bits. Similarly to [6], our decoder algorithm uses only the basic and fast RAM operations such as addition, subtraction, left/right shifts and less-than comparisons. Our scheme avoids the sometimes more costly operations such as multiplication, division or non-standard operations which are pre-computed and stored in a pre-computed table.

Notations. For every two nodes v and w in T , let $P[v, w]$ denote the shortest path connecting v and w in the tree (including v and w), and let $P[v, w) = P[v, w] \setminus \{w\}$.

For two integers $a \leq b$, let $[a, b]$ denote the set of integers $\{a, a+1, \dots, b\}$. We refer to this set as an *interval*. For two intervals $I = [a, b]$ and $I' = [a', b']$, we say that $I \prec I'$ if $b < a'$. The *size* of an interval $I = [a, b]$, denoted by $|I|$, is the number of integers in I , i.e., $|I| = b - a + 1$.

3 Modifying the interval containment test

Our scheme is inspired by the scheme in [8] which was designed for trees of bounded depth. Given a rooted tree T , the label assigned to each node by the scheme in [8] is a pointer to some interval, and an ancestry query between any two given nodes is answered by a simple interval containment test between the corresponding intervals. The underlying idea of that scheme consists in proving that, if T is of small depth, then one can choose the intervals from a small set of intervals U in which the intervals are well nested within themselves. A pointer to an interval in U can be encoded using $\log |U|$ bits, and thus, since U is relatively small, the scheme uses short labels. Unfortunately, this approach is no longer efficient when the tree has long paths. Indeed, in that case, the set U of nested intervals becomes too large.

Informally, enforcing the decoder to be merely an interval containment test imposes a strong constraint on the way the intervals must be organized in U . For arbitrary trees, we could not find a way to bypass this constraint while keeping U small. Instead, we introduce a decoder which, on the one hand, makes the ancestry test somewhat more complicated than when using the interval containment test (yet the test remains very simple), but, on the other hand, enables to organize and nest the intervals in such a way that labels become very small. Our new decoder exploits the

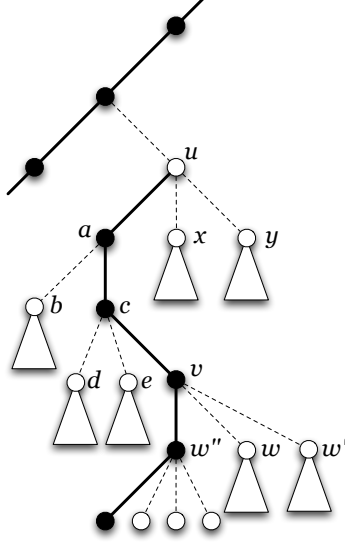


Figure 1: The heavy nodes are depicted in black, while the light nodes are depicted in white. In the figure, $\text{sup}(v) = u$ and $\text{sup}(w) = w$. We have $\text{parent}(w) = v$. Hence, the local quasi ancestors of w are the set of nodes is $\text{lqa}(w) = \{x, y, a, b, c, d, e, v, w'\}$ if $\text{DFS}(w) > \text{DFS}(w')$, and $\text{lqa}(w) = \{x, y, a, b, c, d, e, v\}$, otherwise. Similarly, we have $\text{lqa}(x) = \emptyset$ if $\text{DFS}(y) > \text{DFS}(x)$, and $\text{lqa}(x) = \{y\}$ otherwise.

fact that intervals can be partially ordered not only by the containment relation, but also by the relation \prec introduced in the previous section.

Given any node u of some rooted tree T , we associate u with an interval $I(u)$, and with a *supervisor* node, denoted by $\text{sup}(u)$, which is either u itself or one of its ancestors. For this purpose, we first mark each node as either *heavy* or *light* as follows. For every non-leaf node u of T , let $H(u)$ be the set of children v of u that satisfy $\text{weight}(v) \geq \text{weight}(w)$ for every child w of u . Among the nodes in $H(u)$, select an arbitrary node, and call it *heavy*. A node which is not heavy is called *light*. (In particular, the root is light).

For each node $u \in T$, define the *supervisor* of u , denoted by $\text{sup}(u)$, as the light node of largest depth on the path $P[u, r]$ connecting u to the root r . Note that if u is light then $\text{sup}(u)$ is u itself; in particular, $\text{sup}(r) = r$, and $\text{sup}(\text{sup}(u)) = \text{sup}(u)$. Observe also that if u is an ancestor of v , then either $\text{sup}(u) = \text{sup}(v)$, or $\text{sup}(u)$ is an ancestor of $\text{sup}(v)$. See Figure 1.

As we will show later, the basic rule of our decoder relies on the following definition which is a modification of the interval containment test used in several previous schemes.

Definition 1 *Let us consider a set of intervals $\{I(u), u \in V\}$ for a tree T . We assume that all intervals in the set are distinct, i.e., $I(u) \neq I(v)$ for any two distinct nodes u and v . We say that the decoding conditions hold at u w.r.t. v if and only if*

- D1: $I(v) \subset I(\text{sup}(u))$, and
- D2: $I(u) \prec I(v)$ or $I(u) = I(\text{sup}(u))$.

Given the labels $L(u)$ and $L(v)$ of two nodes in a rooted tree, our boolean decoder \mathcal{D} outputs 1 if and only if the decoding conditions hold at u w.r.t. v . Our marker algorithm will then guarantee the following:

- The decoder is correct, i.e., the intervals associated with each node are selected such that, for any two nodes u and v , the decoding conditions hold at u w.r.t. v if and only if u is an ancestor of v ;
- given a tree T , the interval and labels can be assigned to all nodes in T in linear time;
- given a label $L(u)$, the intervals $I(u)$ and $I(\text{sup}(u))$ can be computed in constant time;
- each label is encoded using $\log n + O(\log \log n)$ bits.

4 The $\log n + O(\log \log n)$ ancestry labeling scheme

We are now ready to prove our main result, that is:

Theorem 1 *There is an ancestry labeling scheme for $\mathcal{T}(n)$ with label size $\lceil \log n \rceil + 6\lceil \log \log n \rceil + 7$ and constant query time. Moreover, given a tree T , the labels can be assigned to the nodes of T in linear time.*

We prove Theorem 1 by constructing an ancestry labeling scheme $(\mathcal{M}, \mathcal{D})$ with the desired properties.

4.1 The marker algorithm \mathcal{M}

For simplicity of presentation assume that n is a power of 2, and let us fix a tree $T \in \mathcal{T}(n)$. Our marker algorithm \mathcal{M} first assigns an interval to each node in a way such that u is an ancestor of v if and only if the decoding conditions hold at u w.r.t. v . For this purpose, we first show that it is sufficient to provide an assignment of intervals that satisfies a more “local” condition.

4.1.1 The local partial order conditions

Let us first assign numbers from 0 to $n - 1$ to the nodes according to a DFS traversal that starts at the root, and visits light children first. We denote by $\text{DFS}(u)$ the DFS number of u . Let $P_u = P[\text{parent}(u), \text{sup}(\text{parent}(u))]$. We define the *local quasi-ancestors* of u , denoted by $\text{lqa}(u)$, as all nodes in P_u , together with their light children, but removing u , $\text{sup}(\text{parent}(u))$, and all nodes that have DFS numbers higher than u . See Figure 1 for an example. Note that the local quasi-ancestors of a node may not form a connected subtree of T .

Definition 2 *Let us consider a set of pairwise distinct intervals $\{I(u), u \in V\}$ for a tree T . For every node u , we say that u satisfies the local partial order (LPO) conditions if the two conditions below are satisfied:*

- LPO1: $I(u) \subseteq I(\text{sup}(u)) \cap I(\text{sup}(\text{parent}(u)))$ for every non root node u ;

- LPO2: $I(x) \prec I(u)$ for every local quasi-ancestor $x \in \text{lqa}(u)$.

Claim 1 *If every node u satisfies the local partial order condition LPO1, then for any light node u , and any descendent v of u , we have $I(v) \subset I(u)$.*

Proof. The claim is established by induction on the distance between u and v . If $\text{dist}(u, v) = 1$ then the claim holds by LPO1. Assume that the claim holds for $\text{dist}(u, v) \leq d$ for $d \geq 1$, and assume $\text{dist}(u, v) = d + 1$. If $\text{sup}(v) = u$ then the claim follows by LPO1. Thus assume $\text{sup}(v) \neq u$. If $\text{sup}(\text{parent}(v)) = u$ then again the claim follows by LPO1. Thus assume also $\text{sup}(\text{parent}(v)) \neq u$. In this case there exists a light node $w \notin \{u, v\}$ on the shortest path connecting u to v . The claim then follows by induction. \square

The following claim relates the decoding conditions to the local partial order conditions.

Claim 2 *If every node u satisfies the local partial order conditions LPO1 and LPO2, then for every two different nodes u and v , u is an ancestor of v if and only if the decoding conditions D1 and D2 hold at u w.r.t. v .*

Proof. Assume that every node u satisfies the local partial order conditions. Consider first the case that u is an ancestor of v . Since v is a descendent of u , either $\text{sup}(v) = \text{sup}(u)$ or $\text{sup}(v)$ is a descendent of $\text{sup}(u)$. Thus, by Claim 1, $I(v) \subseteq I(\text{sup}(v)) \subseteq I(\text{sup}(u))$. Since $v \neq u$, $I(v) \subset I(\text{sup}(u))$, i.e., D1 follows. If u is light then the fact that u and v satisfy D2 follows trivially from the fact that, in this case, $\text{sup}(u) = u$. So assume now that u is heavy. If $u \in \text{lqa}(v)$, then D2 follows from LPO2. Otherwise, if $u \notin \text{lqa}(v)$, then there exists a light node w that is an ancestor of v and such that $u \in \text{lqa}(w)$. D2 follows by combining LPO2 with Claim 1.

Consider now the case that u is not an ancestor of v . We need to show that either D1 or D2 does not hold. Let w be the light node of largest depth on the path $P[v, r]$ which is an ancestor of u . If w is v itself then $\text{sup}(u)$ is either v or a descendant of v . Therefore, by Claim 1, we have $I(\text{sup}(u)) \subseteq I(v)$, and thus D1 is not satisfied. In the remaining proof we thus assume that $w \neq v$.

For a node x which is a descendant of w , and which satisfies $\text{sup}(x) \neq w$, let $f(x)$ be the light node of smallest depth on the path $P[x, w]$. Assume first that $\text{sup}(u) = w$. If also $\text{sup}(v) = w$ then $v \in \text{lqa}(u)$, and thus, by LPO2, $I(v) \prec I(u)$. Since $u \neq w$, we have $I(u) \neq I(\text{sup}(u))$, and therefore D2 is not satisfied. If, on the other hand, $\text{sup}(v) \neq w$ then we have $f(v) \in \text{lqa}(u)$ and $I(v) \subset I(f(v))$. Similarly to the previous case, this implies that D2 is not satisfied.

Assume now that $\text{sup}(u) \neq w$. We have $I(\text{sup}(u)) \subseteq I(f(u))$. If v is an ancestor of $f(u)$ then $v \in \text{lqa}(f(u))$. Consequently, $I(v) \prec I(f(u))$ and thus D1 does not hold. On the other hand, if v is not an ancestor of $f(u)$ then we are left with two cases: $\text{sup}(v) = w$ and $\text{sup}(v) \neq w$. If $\text{sup}(v) = w$ then $I(f(u)) \prec I(v)$ since $f(u) \in \text{lqa}(v)$. Thus D1 does not hold. Finally, if $\text{sup}(v) \neq w$ then either $f(u) \in \text{lqa}(f(v))$ or $f(v) \in \text{lqa}(f(u))$. Since $I(\text{sup}(u)) \subseteq I(f(u))$ and $I(v) \subseteq I(f(v))$, it follows that D1 does not hold. \square

4.1.2 The interval assignment

By Claim 2, one of our goals is to let the marker assign intervals that satisfy the local partial order conditions at each node. For integers a, b and k , let

$$I_{k,a,b} = [2^k a, 2^k(a+b)].$$

For $k \in [1, \log n]$, define the set of intervals:

$$\mathcal{I}_k = \{I_{i,a,b} \mid i \in [1, k], a \in [1, \frac{4n \log n}{2^i}], \text{ and } b \in [1, 4 \log n]\}.$$

Let $\mathcal{I} = \mathcal{I}_{\log n}$.

Definition 3 Let $T \in \mathcal{T}(n)$. We say that a mapping $I : V \rightarrow \mathcal{I}$ is a legal interval-mapping if the mapping is one-to-one, and $\{I(u), u \in V\}$ satisfies the local partial order conditions at each node of T .

In order to show that there exists a legal interval-mapping from every tree in $\mathcal{T}(n)$ into \mathcal{I} , we use the following notation. For any interval $J \subset [1, n \log n]$, and, for any k , $1 \leq k \leq \log n$, let

$$\mathcal{I}_k(J) = \{I_{i,a,b} \in \mathcal{I}_k \mid I_{i,a,b} \subseteq J\}.$$

Lemma 1 For every $k \in [1, \log n]$, every tree $T \in \mathcal{T}(2^k)$, and every interval $J \subseteq [1, 4n \log n]$, such that $|J| = 4k|T|$, there exists an legal interval-mapping of T into $\mathcal{I}_k(J)$. Moreover this mapping can be computed in $O(|T|)$ time.

Proof. We prove the lemma by induction on k . For $k = 1$, the lemma holds trivially. Assume now that the claim holds for k with $1 \leq k < \log n$, and let us show that it also holds for $k + 1$. Clearly, if $|T| \leq 2^k$ then we are done by induction.

Consider now the case where T is of size $2^k < |T| \leq 2^{k+1}$, and let $J \subset [1, 4n \log n]$ be an interval, such that $|J| = 4(k+1)|T|$. Our goal is to show that there exists a legal-interval mapping of T into $\mathcal{I}_{k+1}(J)$.

We make use of the following decomposition of T . Let H be the path from the root of T to a leaf of T such that every non-root node in H is heavy. Let v_1, v_2, \dots, v_d be the nodes of H , ordered top-down, i.e., v_1 is the root of T , v_d is a leaf of T , and for every $1 \leq i < d$, v_i is the parent of v_{i+1} .

For every $1 \leq i \leq d$, let $T_i^1, T_i^2, \dots, T_i^{t_i}$ be the rooted trees hanging down from the light children of v_i . (If v_i does not have any light child, which is the case, for example, for $i = d$, then this set of trees is empty, or, in other words, $t_i = 0$). One important property of these trees is that, for every i and j , $1 \leq j \leq t_i$, we have $|T_i^j| < |T|/2 \leq 2^k$.

We now group the nodes in $T \setminus \{r\}$ in disjoint trees T_1, T_2, \dots, T_m , where $m = (d-1) + \sum_{\ell=1}^d t_\ell$ as follows. A tree T_i is either a single heavy node v_j , for $j > 1$, or a subtree hanging from a light child of some v_j , $j \geq 1$. Moreover, the trees are enumerated according to the DFS numbers of their roots as follows. Recall $\text{DFS}(u)$ denotes the DFS number of node u in T . The trees are ordered such that if r_j denotes the root of T_j then $\text{DFS}(r_j) < \text{DFS}(r_{j+1})$ for all $j = 1, \dots, m-1$. See Figure 2.

Consider now the interval $J \subseteq [1, 4n \log n]$ such that $|J| = 4(k+1)|T|$, and express it as $J = [\alpha, \alpha + 4(k+1)|T| - 1]$ for some integer $\alpha \leq 4n \log n$. Let a be the smallest integer such that $\alpha \leq a 2^{k+1}$, and let b be the smallest integer such that $4k|T| \leq b 2^{k+1}$.

First, we assign the root r to the interval $J' = [a 2^{k+1}, (a+b) 2^{k+1}]$. We now show that indeed $J' \in \mathcal{I}_{k+1}(J)$. By definition of a and b , we have

$$(a+b)2^{k+1} = 2^{k+2} + ((a-1) + (b-1))2^{k+1} \leq 2^{k+2} + \alpha + 4k|T| - 2.$$

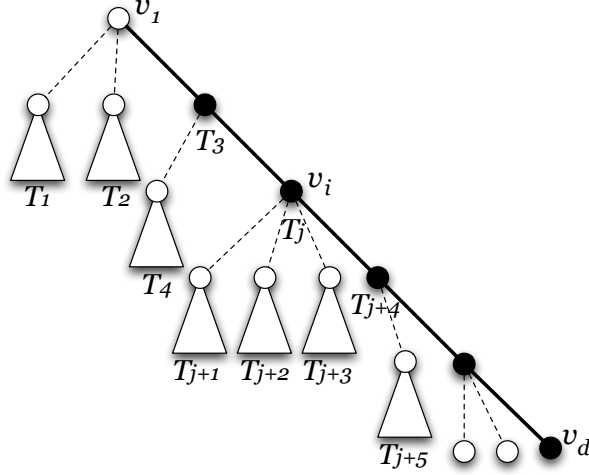


Figure 2: Tree decomposition as in the proof of Lemma 1. In the figure, the DFS traversal that visits light nodes first is supposed to proceed by visiting children from left to right.

Since $2^k < |T|$, we get that

$$(a + b)2^{k+1} < \alpha + (4k + 4)|T| - 1.$$

Thus

$$J' = [a2^k, (a + b)2^{k+1}] \subset [\alpha, \alpha + (4k + 4)|T| - 1] = J.$$

Therefore, since $1 \leq a \leq \alpha/2^{k+1} \leq 4n \log n/2^{k+1}$, and $1 \leq b \leq 4 \log n$, we obtain that $J' \in \mathcal{I}_{k+1}(J)$.

The rest of the nodes in T are mapped as follows. First note that $|J'| \geq 4k|T|$, and recall that $\sum_{i=1}^m |T_i| = |T| - 1$. We break J' into $m + 1$ consecutive intervals $J'_1, J'_2, \dots, J'_{m+1}$ such that, for each $1 \leq i \leq m$, we have $|J'_i| = 4k|T_i|$ and $J'_i \prec J'_{i+1}$. For each $1 \leq i \leq m$, since $|T_i| \leq 2^k$, we can use the induction hypothesis to map the nodes in T_i to $\mathcal{I}_k(J'_i)$ via a legal interval-mapping.

The fact that the above recursive mapping can be performed in linear time is obvious. It remains to show that the above mapping of T into $\mathcal{I}_{k+1}(J)$ is indeed a legal interval-mapping. That is, we have to show that the set of intervals satisfies the local partial order conditions LPO1 and LPO2 at each node u . The conditions hold trivially at the root r of T . The fact that the conditions hold for every node u in $T_i \setminus r_i$, follows from the induction hypothesis, and because both $\text{sup}(u)$, $\text{sup}(\text{parent}(u))$, and $\text{lqa}(u)$ are all contained in T_i . Finally, consider the root r_i of T_i . (Note that if r_i is heavy then $T_i = \{r_i\}$). LPO1 holds trivially for r_i because J' contains J_i , and, by induction, the interval assigned to r_i is contained in the interval $J'_i \subset J'$. To establish that LPO2 holds, first observe that $\text{lqa}(r_i) = \{r_1, \dots, r_{i-1}\}$. On the other hand, for every $j = 1, \dots, i - 1$, $\text{DFS}(r_j) < \text{DFS}(r_i)$, and thus $J'_j \prec J'_i$. Hence LPO2 holds for r_i as well. Our mapping is thus a legal interval-mapping of T into $\mathcal{I}_{k+1}(J)$. This completes the proof of the lemma. \square

By taking $k = \log n$ in the above lemma, we obtain the following.

Corollary 1 *Let $T \in \mathcal{T}(n)$. There exists a legal interval-mapping of T into $\mathcal{I}_{\log n}([1, 4n \log n]) \subset \mathcal{I}$.*

4.1.3 The label assignment

We are now ready to describe the label $L(u)$ assigned to every node u by our marker algorithm \mathcal{M} . Given a rooted tree $T \in \mathcal{T}(n)$, the marker \mathcal{M} first marks each node as either heavy or light, and then assigns the DFS numbers. This clearly takes linear time. Then the marker maps the nodes of T into \mathcal{I} using the legal-interval mapping given in Corollary 1. Again, this step takes linear time.

Given a node u , the marker uses the first $\log n + 3\lceil \log \log n \rceil + 3$ least significant bits of $L(u)$ to encode the interval $I(u)$. This can be done explicitly as $I(u)$ is of the form $I_{i,a,b}$ for some $i \in [1, \log n]$, $a \in [1, \frac{4n \log n}{2^i}]$ and $b \in [1, 4 \log n]$.

Finally, the marker aims at encoding $I(\text{sup}(u))$ in the label of u . However, using the method above to encode $I(\text{sup}(u))$ would consume yet another $\log n + 3\lceil \log \log n \rceil + 3$ bits, which is obviously undesired. Instead, we use the following trick. Let i', a' , and b' be such that $I(\text{sup}(u)) = I_{i',a',b'}$. (Note that if $\text{sup}(u) = u$ then we simply have $i' = i$, $a' = a$ and $b' = b$). Clearly, $2\lceil \log \log n \rceil + 2$ bits suffice to encode both i' and b' . To encode a' , the marker acts as follows. Let $I(u) = [\alpha, \beta]$, and let a'' be the largest integer such that $2^{i'} a'' \leq \alpha$. Recall that by definition $I(\text{sup}(u)) = [2^{i'} a', 2^{i'}(a' + b')]$. We have, $a'' - 4 \log n \leq a'' - b'$ because $b' \leq 4 \log n$. Since $I(u) \subseteq I(\text{sup}(u))$, we also have $2^{i'}(a' + b') \geq \beta > \alpha \geq 2^{i'} a''$. Thus $a'' - b' < a'$. Finally, again since $I(u) \subseteq I(\text{sup}(u))$, we have $2^{i'} a' \leq \alpha$, and thus $a'' \geq a'$. Combining the above inequalities, we get that $a' \in [a'' - 4 \log n - 1, a'']$. The marker now encodes the integer $t \in [0, 4 \log n - 1]$ such that $a' = a'' - t$. This is done in consuming another $\lceil \log \log n \rceil + 2$ bits. Hence, the following follows by construction:

Lemma 2 *Given a tree $T \in \mathcal{T}(n)$, the marker \mathcal{M} assigns labels to the nodes of T in linear time, and each label is encoded using $\log n + 6\lceil \log \log n \rceil + 7$ bits.*

4.2 The decoder \mathcal{D}

Now, we describe our decoder \mathcal{D} . Given the labels $L(u)$ and $L(v)$ assigned by \mathcal{M} to two different nodes in some tree T , the decoder \mathcal{D} needs to find whether u is an ancestor of v in T . (Observe that since each node receives a distinct label, the decoder can easily find out if u and v are in fact the same node, and, in this trivial case, it simply outputs 0.)

The decoder inspects the first $\log n + 3\lceil \log \log n \rceil + 3$ least significant bits of $L(u)$ to extract $I(u)$ (recall that $I(u) = I_{i,a,b}$ is encoded by storing explicitly the three parameters i , a , and b). Then, once $I(u) = [\alpha, \beta]$ has been reconstructed from $L(u)$, the decoder aims at extracting $I(\text{sup}(u))$. For this purpose, it first reconstructs i' and b' that have been explicitly encoded in the next $2\lceil \log \log n \rceil + 2$ bits. Then, it computes the largest integer a'' such that $2^{i'} a'' \leq \alpha$. The decoder then proceeds by extracting t , and computes $a' = a'' - t$. At this point, \mathcal{D} have reconstructed both $I(u)$ and $I(\text{sup}(u))$. Similarly, \mathcal{D} extracts $I(v)$ by inspecting $L(v)$.

Finally, the boolean decoder \mathcal{D} outputs 1 if and only if the two decoding conditions D1 and D2 hold at u w.r.t. v (see Definition 1).

Lemma 3 *Let $L(u)$ and $L(v)$ be two labels assigned by \mathcal{M} to two nodes in T . The decoder $\mathcal{D}(L(u), L(v))$ performs in constant time, and satisfies $\mathcal{D}(L(u), L(v)) = 1$ if and only if u is an ancestor of v in T .*

Proof. The fact that $\mathcal{D}(L(u), L(v)) = 1$ if and only if u is an ancestor of v in T follows from the fact that the intervals are assigned by the marker via a legal-interval mapping (cf. Corollary 1).

Since $I(u) = I_{i,a,b} = [2^i a, 2^i(a+b)]$ with all three parameters i , a , and b stored explicitly, computing $I(u)$ from $L(u)$ can be achieved in constant time. (Note that $2^i a$, for example, can be obtained from a but a simple shift of i bits.) Similarly, $I(v)$ can be extracted from $L(v)$ in constant time. Computing $I(\text{sup}(u))$ just needs a simple subtraction and a division by a power of 2, which again amounts to a simple shift operation. The lemma follows. \square

This completes the proof of Theorem 1.

5 Conclusion

Our ancestry labeling scheme is using labels of optimal size $\log_2 n + O(\log \log n)$ bits, to the price of a decoding mechanism based of an interval condition slightly more complex than the simple interval containment condition. Although this has no impact on the decoding time (our decoder still works in constant time), the question of whether there exists an ancestry labeling scheme with labels of size $\log_2 n + O(\log \log n)$ bits, but using solely the interval containment condition, is intriguing.

Acknowledgments: the authors are very thankful to Sundar Vishwanathan and Jean-Sebastien Sereni for helpful discussions.

References

- [1] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo and T. Rauhe. Compact labeling schemes for ancestor queries. *SIAM Journal on Computing* **35**, (2006), 1295–1309.
- [2] Abiteboul, S., Kaplan, H., and Milo, T.: Compact labeling schemes for ancestor queries. In Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2001.
- [3] S. Alstrup, P. Bille and T. Rauhe. Labeling Schemes for Small Distances in Trees. In Proc. 14th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2003.
- [4] S. Alstrup, C. Gavoille, H. Kaplan and T. Rauhe. Nearest Common Ancestors: A Survey and a new Distributed Algorithm. *Theory of Computing Systems* **37**, (2004), 441–456.
- [5] S. Alstrup and T. Rauhe. Small induced-universal graphs and compact implicit graph representations. In Proc. 43rd IEEE Symp. on Foundations of Computer Science (FOCS), 2002.
- [6] S. Alstrup and T. Rauhe. Improved labeling scheme for ancestor queries. In Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 947-953, 2002.
- [7] P. Fraigniaud and C. Gavoille. Routing in trees. In Proc. 28th Int. Colloq. on Automata, Languages & Prog. (ICALP), LNCS 2076, pages 757–772, Springer, 2001.
- [8] P. Fraigniaud and A. Korman. Compact Ancestry Labeling Schemes for XML Trees. In Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA), 2010.
- [9] C. Gavoille and D. Peleg. Compact and Localized Distributed Data Structures. *Distributed Computing* **16**, (2003), 111–120.

- [10] C. Gavaille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. In Proc. 12th ACM-SIAM Symp. on Discrete Algorithms (SODA), pages 210–219, 2001.
- [11] M. Katz, N.A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM Journal on Computing* **34** (2004), 23–40.
- [12] A. Korman. Labeling Schemes for Vertex Connectivity. *ACM Transactions on Algorithms*, to appear.
- [13] A. Korman. Improved Compact Routing Schemes for Dynamic Trees In Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC), 2008.
- [14] A. Korman and S. Kutten. Distributed Verification of Minimum Spanning Trees. *Distributed Computing* **20**(4): 253–266 (2007).
- [15] H. Kaplan, T. Milo and R. Shabo. A Comparison of Labeling Schemes for Ancestor Queries. In Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002.
- [16] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. on Discrete Math* **5**, (1992), 596–603.
- [17] D. Peleg. Informative labeling schemes for graphs. In Proc. 25th Symp. on Mathematical Foundations of Computer Science (MFCS), LNCS 1893, pages 579–588. Springer, 2000.
- [18] N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal* **28**, (1985), 5–8.
- [19] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. of the ACM* **51**, (2004), 993–1024.
- [20] M. Thorup and U. Zwick. Compact routing schemes. In Proc. 13th ACM Symp. on Parallel Algorithms and Architecture (SPAA), pages 1–10, 2001.